The Privilege and Responsibility of Software Development

Grady Booch
Rational Software Corporation

As professionals, it is a tremendous privilege to be part of an industry that delivers software that makes a fundamental difference to our organizations, our country, and our civilization. At the same time, however, we must realize that creating quality software that matters is intrinsically hard. As such, as professionals, we have a deep responsibility to do our work with purpose, courage, and a sense of moral purpose.

No one really wants software. End users typically hate software for many reasons. It is that thing that gets in the way of their work (by driving unnatural work processes). It wastes time (when their machines go down), distracts them (by offering up a deluge of useless or misleading information), and generally annoys them (when the bones of the underlying implementation show through). Users simply want to accomplish their mission, and insofar as any underlying software makes its presence known, it is counter to getting the job done.

Program managers often hate software as well. It is that thing that eats up budgets with an insatiable appetite for growth. It is terribly slippery to get ones hands around, and even if you do, it has a tendency to slip out of your control at a moment's notice leaving an ugly, smelly mess on the floor.

Bad software – and there is far too much of it in the world – not only wastes time and budgets, distracts, and annoys, it can also put lives, businesses, and whole economies at risk [1]. Even at its best, a software-intensive system can amplify human intelligence, but it cannot replace human judgment; a software-intensive system can fuse, coordinate, classify, and analyze information, but it cannot create knowledge.

Still, we bring software into our lives and into our systems for some very basic reasons. There are some things that we can do in software that we cannot do otherwise:

- Control an aerodynamically unstable aircraft.
- Fuse and analyze information from a multitude of sensors so as to form a unique view of the world.
- Create virtual worlds wherein experiments that would otherwise be too dangerous to conduct can be carried out.
- Search through terabytes of information in the beat of a heart.

Furthermore, software offers greater flexibility than can be offered in hardware, which is why the mix of software to hardware within many systems is growing in favor of software. Finally, for the most part, investment in software has an undeniable economic return: Across the spectrum, from embedded systems to command and control systems to enterprise information systems, the presence of software adds essential value, far more than the investment necessary to create that software.

As I look back over the history of software development, it strikes me that ours is an industry that has largely grown out of demand from users who want more from their systems for less. All of our systems are constrained by the laws of physics and by a few laws of software [1]. But for the most part, it is our ability as an industry to develop better software faster, software that meets the needs of its end users, that is the primary constraint upon meeting our vision for what software can do in the world. Insofar as a given software development team can execute well, they enable the mission for which they labor; insofar as they execute inefficiently or not at all, they fail the organization that commissioned them.

> *"... a software-intensive system can amplify human intelligence, but it cannot replace human judgment ..."*

In that sense, software development is – or certainly should be – considered an engineering discipline. From the perspective of a software-intensive project, there exists the competing forces of cost, schedule, functionality, compatibility, performance, capacity, scalability, reliability, availability, security, fault tolerance, and resilience, all in the presence of technology and business churn. Balancing these forces is very much an engineering activity. There is no such thing as a *perfect* design or a *perfect* system; indeed, the very presence of any new system changes the way its stakeholders view the world and thus alters their vision for what that new system should have done in the first place.

We as developers are the ones who do the heavy lifting, creating, and rearranging the components that make up our software worlds to form systems that balance these forces.

As developers, we have all had our share of bad days: days that our operating systems, networks, workstations, and co-workers conspire against us to suck all productivity out of the air; days that our bosses or their bosses or our customers hammer us for errors done or for functions left undone; or days that turn into nights and back into days again as we chase some elusive gnome from our system.

These are the days of living as a netslave [2], a microserf [3]. After abiding such days during which we labor to build artifacts that live in the realm of nanoseconds, sometimes we long for a life with "no unit of time shorter than a season" [4].

Still, most of us come to the profession of software development deliberately, typically because we like to create things from pure thought, things that give life to our machines and that matter to our organizations, perhaps even to the world. For others, software creeps up behind us and grabs us by the neck; although we may secure an uneasy truce with it even though we may not be code warriors, we still require some degree of development skills so that we can wrestle that software to the ground and direct it to carry out our will. Either way, as an intentional or as an accidental developer, we build things that the rest of the world needs and uses and yet are often invisible to them.

For this reason, it is both a privilege as well as a deep responsibility to be a software developer.

It is a privilege because – in spite of some inevitable dark days – we collectively are given the opportunity to create things that matter: to individuals, to teams, to organizations, to countries, to our civilization. We have the honor of delivering the

stuff of pure intellectual effort that can protect, defend, heal, serve, entertain, connect, and liberate, freeing the human spirit to pursue those activities that are purely and uniquely human.

Paul Levy, Rational's chairman, once noted the following:

> Ultimately, building software [is] the world's most important industry. Software today allows a brother in San Jose to call a sister in St. Petersburg. Software today speeds the process of drug discovery, potentially curing Alzheimer's. Software today drives the imaging systems that allow the early detection of breast cancer and other maladies. Software controls the passive restraint systems and anti-lock breaking systems that save children's lives in automobiles every day. Software powers our communications and transportation technologies. Software allows us to peer deep within ourselves and study the human genome. Software allows us to explore and understand our universe. And, make no mistake about it, we are just getting started. [5]

Simultaneously, we have a deep responsibility. Because individuals and organizations depend on the artifacts we create, we have an obligation to deliver quality systems using scarce human and computing resources intentionally and wisely. This is why we hurt when our projects fail, not only because each failure represents our inability to deliver real value, but also because life is too short to spend precious time on constructing bad software that no one wants, needs, or will ever use.

As professionals, we also have a moral responsibility: Do we choose to labor on a system that we know will fail or that might steal from another person their time, their liberty, or their life? Questions like this have no technical answers, but rather are ones that must be consciously weighed by our individual belief system as we deploy technology to the world.

At the very least, the consequences of our failure may be as simple as the delivery of annoying software, which behaves in unexpected ways or is so fragile that it drives the user rather than letting the user drive it. Such software wastes our time and gets in the way of accomplishing real work. At the other extreme, the consequences may be life threatening: The software fails, and people die.

Across this entire spectrum of bad software, it is partly a failure of the team in the sense that the *organization* failed to deliver a useful system that worked. However, it is ultimately a failure of the *individual*: Denying all responsibility by hiding inside an organization is no excuse for this kind of failure.

Personal responsibility can manifest itself in a variety of ways: arguing against unrealistic schedules, working out of the box where it might yield a solution that sidesteps the current barriers to progress, expecting quality, and demanding the best from your colleagues. To do otherwise permits an environment in which a succession of lies is permitted to flourish, with the inevitable delivery of bad software.

Thus, software development is ultimately a human activity, not only because it emanates from the human intellect, but also because it requires the cooperative activity of others to make it real.

As professionals, we therefore constantly seek better ways to deliver quality software that matters, because the task is too complex to squander our time and our energy. This is why we analyze *why* projects were successful and similarly look at failed projects to learn from their mistakes. We then codify all these lessons learned in the best practices and processes that constitute our industry's tribal memory, such as found in the Rational Unified Process and in emerging ideas from eXtreme Programming. For the same reason, we agree upon common notations such as the Unified Modeling Language that help us communicate and reason about our systems.

Still, the demand for software continues to rise at a staggering rate. The ever-growing capabilities of hardware combined with increasing social awareness and economic value of the utility of computers create tremendous pressure to automate systems of even greater complexity. Thus, our privilege to carry out our skills continues, as well as does our responsibilities.

Indeed, as Levy said, "We are just getting started." Software is perhaps the most splendid material to build things: We create software from pure thought and shape it at will to form new worlds limited only by our imagination. As professionals, we labor to build quality systems that are useful and that work. As software engineers, we face the task of creating complex systems with elegance in the presence of scarce computing and human resources. Inescapably, economic realities demand that we build such systems purposefully and efficiently. Developing quality software that matters is fundamentally hard; ultimately, however, our rewards are great, for what we do as an industry changes the world.

And that is why I am – as we all should be – both proud and humbled to be called a software developer.◆

References

1. Harel, D. Computers, Ltd.: What They Really Can't Do. London: Oxford University Press, 2001.
2. Lessard, B., and S. Baldwin. NetSlaves: True Tales of Working the Web. New York: McGraw-Hill, 2000.
3. Coupland, D. microserfs. New York: HarperCollins, 1995.
4. Kidder, T. The Soul of a New Machine. New York: Atlantic-Little, Brown Books, 1981: 220.
5. Levy, Paul. Keynote Address. Rational User Conference. Sept. 2001.

Note

1. See "ACM Forum on Risks to the Public in Computers and Related Systems." Peter Neumann moderator. <http://catless.ncl.ac.uk/Risks>.

About the Author

Grady Booch, is chief scientist at Rational Software Corporation and has been since its inception in 1980. He is recognized internationally for his innovative work on software architecture, modeling, and software engineering. Booch is one of the original developers of the Unified Modeling Language (UML) and was also one of the original developers of several of Rational's products. He is the author of six best-selling books, including the "UML User Guide" and the seminal "Object-oriented Analysis and Design with Applications" and has published several hundred technical articles on software engineering. He is also an ACM Fellow and a Rational Fellow as well as a board member of the Agile Alliance and the Hillside group. Booch has a bachelor's of science degree in engineering from the United States Air Force Academy and a master's of science degree in electrical engineering from the University of California at Santa Barbara.

Rational Software Corporation
6533 West Prentice Avenue
Littleton, CO 80123
Phone: (303) 986-2405
E-mail: egb@rational.com